Kinematics Model

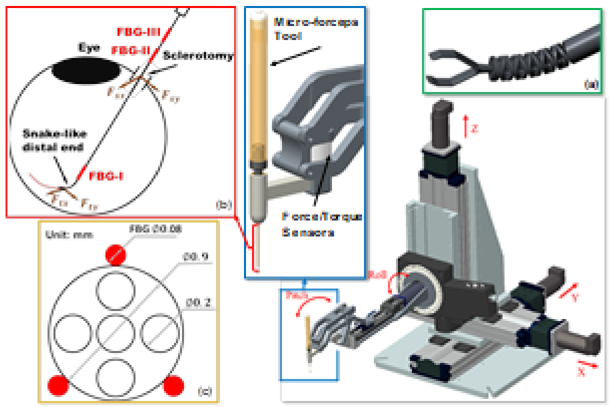

Figure 1: Overview of SHER and I^{2}RIS

1. Forward Kinematics of SHER
   SHER has a total of 5 joints, including x,y and z translations and rotations around y-axis and x-axis. These joints are called q_1 to q_5 respectively. We will use the notation $F = [R, p]$ for transformation and $Rot(axis, angle)$ for rotation.

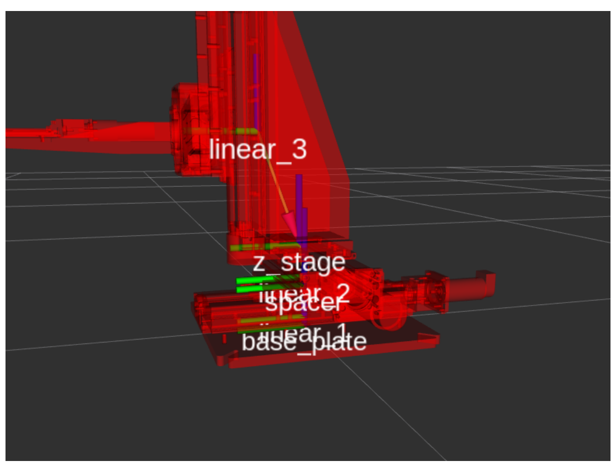

Figure 2: Frames including X and Y Stages

From "base_plate" to "linear_1", $F_1 = [\mathbb{I}, (0, 0, 0.0127)^T]$
From "linear_1" to "spacer", $F_2 = [\mathbb{I}, (0, q_1, 0.04725)^T]$
From "spacer" to "linear_2", $F_3 = [\mathbb{I}, (0, 0, 0.0127)^T]$
From "linear_2" to "z_stage", $F_4 = [\mathbb{I}, (q_2, 0.0075, 0.04725)^T]$
From "z_stage" to "linear_3", $F_5 = [\mathbb{I}, (0, 0.058, 0.15858)^T]$

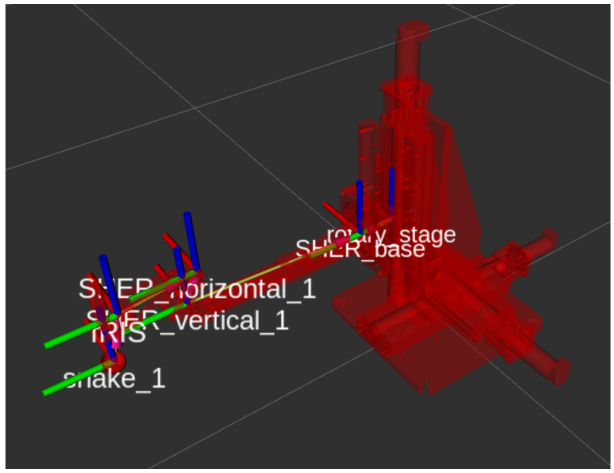

Figure 3: Frames including Z Stages, Roll and Pitch

From "linear_3" to "rotary_stage", $F_6 = [\mathbb{I}, (00.04725, q_3)^T]$
From "rotary_stage" to "SHER_base", $F_7 = [Rot((0, 1, 0), q_4), (0, 0.063, 0)^T]$
From "SHER_base" to "SHER_horizontal_1", $F_8 = [Rot((1, 0, 0), q_5), (0, 0.304, 0.015)^T]$
From "SHER_horizontal_1" to "SHER_vertical_1", $F_9 = [Rot((1, 0, 0), -q_5), (0, -0.023, 0.048)^T]$
From "SHER_vertical_1" to "IRIS", $F_{10} = [Rot((1, 0, 0), q5), (0, 0.120, -0.015)^T]$
From "IRIS" to "Snake_1", $F_{11} = [\mathbb{I}, (0, 0.02177, -0.07628)^T]$
From "Snake_1" to "1_2"(first virtual snake joint), $F_{12} = [\mathbb{I}, (0, 0.02177, -0.07628)^T]$
The forward kinematics of SHER is then multiplication of F_1 to F_{12}.

2. Forward Kinematics of I^2RIS

   I^2RIS has two input joint angles named q_6 and q_7. These two angles control the amount and direction of rotation between each two links of the snake. Note that the direction of rotation alternates from link to link, and is perpendicular to the last one. q_6 represents the rotation around the y-axis, while q_7 represents the rotation around the x-axis.

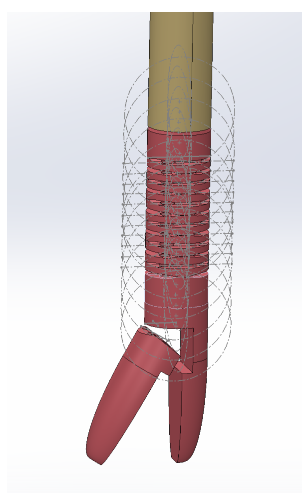

Figure 4: Snake End of I^2RIS

There is a spherical face between each two link of the snake. We can construct two virtual circles as shown in Figure 5, which fits the spherical surfaces, to represent rotation between links. We denote the rotation around y-axis $R_6 = Rot((0,1,0), q6)$ and the rotation around x-axis $R_7 = Rot((1,0,0), q7)$.

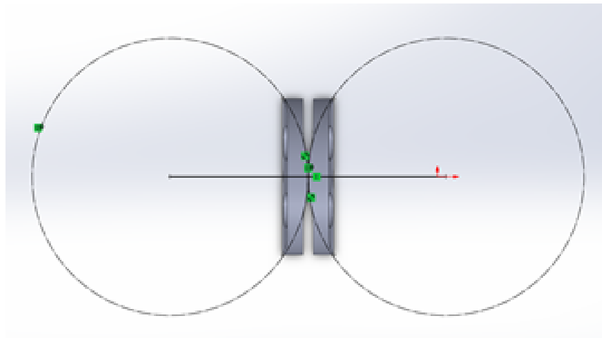

Figure 5: Joint Mechanism of Snake Distal End

The transformation between any two links could be represented as two transformation matrices with the same rotation part (either R_6 or R_7) such as $F_{7a} = [R_7, (0,0,0.00145)^T]$ and $F_{7b} = [R_7, (0,0,-0.0016)^T]$. The forward kinematics of the snake would include the multiplication of 12 pairs of such transformation matrices with the first one being $[R_6, (0,0,0)^T]$, which is then postmultiplied by $[\mathbb{I}, (0,0,-0.00195)^T]$.

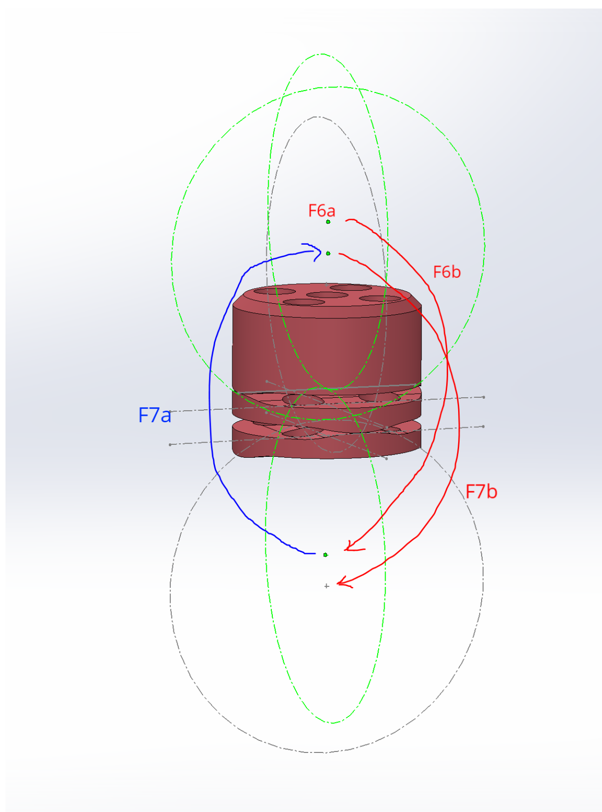

Figure 6: Frame Transformation in the Snake Robot

3. Forward Kinematics from Eye Origin to the First Virtual Snake Joint
   Given the rotations around y-axis and x-axis and insertion distance of I^2RIS, we can find out the transformation matrix between the eye origin and the first virtual snake joint $F = F_1 \cdot F_2 \cdot F_3$. And $F_1 = [Rot((0,1,0), q4), (0,0,0)^T]$, $F_2 = [Rot((1,0,0), q5), (0,0,0)^T]$ and $F_3 = [\mathbb{I}, (0,0,-insertion\,distance)^T]$.

4. Inverse Kinematics

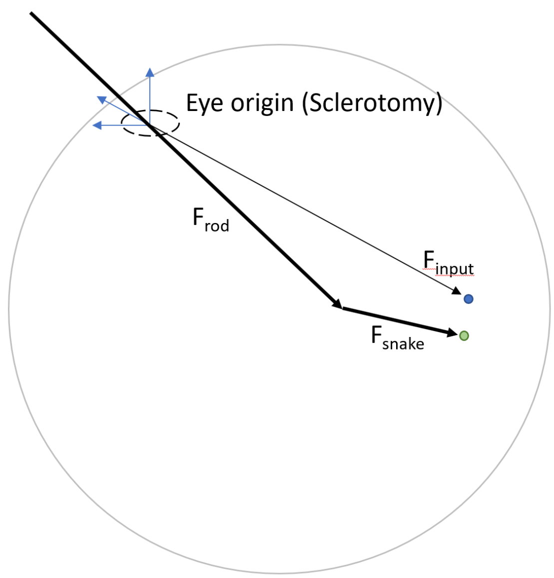

Figure 7: Kinematics Model including the Eye and Sclerotomy

To control the robot and surgical tool in the eyeball, we are given the frame of eye origin and the frame of the goal position of snake tip F_{input}. Since the forward kinematics of the snake contains high order terms, it is difficult to solve for inverse kinematics analytically. Therefore, we choose a numerical solver to solve for inverse kinematics from eye origin (sclerotomy) to tool tip using gradient descent and an analytical solver to solve for the rest of the joints.

Algorithm 1: InvKinSolver

Input: p_{goal} in terms of x,y,z,alpha,beta,gamma and q_{curr}
Output: q_{goal}
error = some large number
while $\Delta x \geq$ *error threshold* **do**

   $F_{rod} = f(dist, roll_{rod}, pitch_{rod})$
   $F_{snake} = f(pitch_{snake}, yaw_{snake})$
   $F_{eye} = F_{rod} \cdot F_{snake}$
   $error\ \Delta x = goalposition - currentposition$
   $pinv(Jacobian(q_{curr})) \rightarrow InvJacobian$
   $InvJacobian \cdot (\alpha \cdot \Delta x) \rightarrow \Delta q$
   $q_{curr} + \Delta q \rightarrow q_{curr}$
   $q_{goal} - q_{curr} \rightarrow \Delta x$

Appendices

```
function F = FwdKin_Base_RodTip(q1,q2,q3,q4,q5)
    % generated from joints of robot URDF
    F1 = getF(eye(3),[0 0 0.0127]);  % translate up to linear_y
    F2 = getF(eye(3),[0 q1 0.04725]);   % q1 joint (y)
    F3 = getF(eye(3),[0 0 0.0127]);
    F4 = getF(eye(3),[q2 0.0075 0.04725]);  % q2 joint (x)
    F5 = getF(eye(3),[0 0.058 0.15858]);
    F6 = getF(eye(3),[0 0.04725 q3]);   % q3 joint (z)
    F7 = getF(Rot([0 1 0],q4),[0 0.063 0]);  % q4 joint (roll)
    F8 = getF(Rot([1 0 0],q5),[0 0.304 0.015]);   % q5 joint (pitch)
    F9 = getF(Rot([1 0 0],-q5),[0 -0.023 0.048]);  % rotating vertical link to horizontal link
    F10 = getF(Rot([1 0 0],q5),[0 0.120 -0.015]);  % rotating horizontal link to IRIS
    F11 = getF(eye(3),[0 0.02177 -0.07628]);  % -0.00007 is ?????
    F12 = getF(eye(3),[0 0 0.00065]);  % from center of snake_1 to start of first virtual snake joint

    F = F1*F2*F3*F4*F5*F6*F7*F8*F9*F10*F11*F12;
end
```

```
function F = FwdKin_RodTip_SnakeTip(q6, q7)
    axis_6 = [0 1 0];
    axis_7 = [1 0 0];
    R6 = Rot(axis_6,q6);
    R7 = Rot(axis_7,q7);

    %starts at first virtual joint (q6)
    F6 = getF(R6,[0 0 0]);
    F6a = getF(R6,[0 0 -0.0016]);
    F7 = getF(R7,[0 0 0.00145]);
    F7a = getF(R7,[0 0 -0.0016]);
    F6b = getF(R6,[0 0 0.00145]);
    F6c = getF(R6,[0 0 -0.0016]);
    F7b = getF(R7,[0 0 0.00145]);
    F7c = getF(R7,[0 0 -0.0016]);
    F6d = getF(R6,[0 0 0.00145]);
    F6e = getF(R6,[0 0 -0.0016]);
    F7d = getF(R7,[0 0 0.00145]);
    F7e = getF(R7,[0 0 -0.0016]);
    F6f = getF(R6,[0 0 0.00145]);
    F6g = getF(R6,[0 0 -0.0016]);
    F7f = getF(R7,[0 0 0.00145]);
    F7g = getF(R7,[0 0 -0.0016]);
    F6h = getF(R6,[0 0 0.00145]);
    F6i = getF(R6,[0 0 -0.0016]);
    F7h = getF(R7,[0 0 0.00145]);
    F7i = getF(R7,[0 0 -0.0016]);
    F6j = getF(R6,[0 0 0.00145]);
    F6k = getF(R6,[0 0 -0.0016]);
    F7j = getF(R7,[0 0 0.00145]);
    F7k = getF(R7,[0 0 -0.0016]);
    F_tip = getF(eye(3),[0 0 -0.00195]);
    %ends at rotation of joint q7k

    FA = F6*F6a*F7*F7a*F6b*F6c*F7b*F7c;
    FB = F6d*F6e*F7d*F7e*F6f*F6g*F7f*F7g;
    FC = F6h*F6i*F7h*F7i*F6j*F6k*F7j*F7k*F_tip;

    if isnumeric(q6)
        F = FA*FB*FC;
    else
        F = simplify(FA)*simplify(FB)*simplify(FC);
    end
end
```

```
function F = FwdKin_EyeOrigin_RodTip(roll,pitch,dist)
    %virtual kinematics of the IRIS rod that is within the eye
    %INPUT: roll pitch insertion_distance
    %output: foward transformation

    F1 = getF(Rot([0 1 0],roll),[0 0 0]);  % roll
    F2 = getF(Rot([1 0 0],pitch),[0 0 0]);   % pitch
    F3 = getF(eye(3),[0 0 -dist]);

    F = F1*F2*F3;
end
```

```
classdef InvKinSolver

    properties
        JacobianObj
    end

    methods
        function obj = InvKinSolver()
            obj.JacobianObj = Jacobians();
        end
        function q = InvKin(obj,pose_goal,q_curr)
            % INPUT: pose_goal is x,y,z,alpha,beta,gamma
            %        q_curr is the current joint value
            % OUTPUT: q_goal
            alpha = 0.5;
            error = 1;  %some large initial value
            q = q_curr;
            xyz_goal = transpose(pose_goal(1:3)); %the goal pos
            %J = Jacobians();
            while error > 0.00005 %0.01 mm error
                %disp("error: " + num2str(error));
                F = InvFwdKin_RCM_EyeOrigin(q(1),q(2),q(3))*FwdKin_RCM_RodTip(q(4),q(5))*FwdKin_RodTip_SnakeTip(q(6),q(7));
                xyz = round(F(1:3,4),8);   %current position
                del = round(xyz_goal-xyz,8);
%                 disp("xyz_goal");
%                 disp(transpose(xyz_goal));
%                 disp("xyz_curr");
%                 disp(transpose(xyz));
%                 disp("del");
%                 disp(del);
                error = double(round(norm(del),5));
                dist = round(0.02763-norm(q(1:3))*sign(q(3)),8);
                dist = min(dist,0.02763);   %dist can't more than the length of the rod to RCM point
%                 disp("dist");
%                 disp(dist);
                deltaX = alpha*(del);
%                 dist q(5) q(4) q(7) q(6)
                dQ = transpose(obj.JacobianObj.EyeInvLookUp([q(4) q(5) dist q(6) q(7)])*deltaX) ; %get [dRoll dPitch dDist dSnakeYaw dSnakePitch]
                dQ = round(dQ,8);
%                 disp("dQ");
%                 disp(dQ);
                dist_new = round(dist + dQ(3),8);

                deltaQ = [0 0 0 dQ(1) dQ(2) dQ(4) dQ(5)];
%                 disp("deltaQ:");
%                 disp(deltaQ);
                q = double(q+deltaQ);

                linear = FwdKin_EyeOrigin_RCM(q(4),q(5),dist_new);  %backsolve for linear motors
                q(1:3) = [linear(2,4) linear(1,4) linear(3,4)];
%                 disp("curr q");
%                 disp(q);
                q = q_limits(q);  %% Insert constraints here
            end
        end
    end
end
```

References